Timers (continued)

Profs. Karkal Prabhu and Naga Kandasamy
ECE Department, Drexel University

Complete the enclosed problem and submit via BBLearn. You will find detailed submission instructions towards the end of this document. Your code must be clearly written, properly formatted, and well commented for full credit.

Online Monitoring of Program Execution

(20 points) A *watchdog timer* (*WDT*) is a hardware timer that automatically generates a system reset if the main program neglects to periodically service it. The WDT is often used to automatically reset an embedded device that hangs because of a software or hardware fault. The WDT_A module can be configured to perform the watchdog function on the MSP432 board. If the timer is not needed in your application, it can be disabled or be configured as an interval timer that can generate interrupts at selected time intervals.

Consider the following code snippet belonging to a safety-critical embedded application that executes a function called `foo()`.

```
int main(void)
{
    while (1) {
        foo(params);
    }
}
```

We would like to monitor `foo's` execution at run time to detect if the function hangs due to a hardware fault or a software bug. If so, we would like to reset the system and restart execution. If the fault is *transient* in nature, such as electromagnetic interference that corrupts a state variable in the program, then reexecution will usually fix the problem. *Permanent faults* such as a design bug in the software are harder to tolerate once the system is deployed in the field. In such cases, we hope that the program state that caused the design fault to manifest itself occurs infrequently over the system's lifetime. Each time this specific state triggers the fault, we recover by restarting the system. Using a WDT for execution monitoring requires a profiling step. Here, we measure the execution time incurred by `foo()` for various input parameters `params`. This is an imprecise step since we may only be able to test `foo` for a subset of `params`. Exhaustively testing all possible combinations of input parameters, especially those with floating-point values, may be too time consuming or impossible. The function `foo` is therefore tested using a representative set for `params`. The maximum measured execution time, also called *worst-case execution time* or WCET, is used to set the timeout interval for the WDT. Suppose we obtain a WCET of three seconds for `foo`, the WDT can be set to timeout if not serviced within four seconds.

The following code snippet shows how to configure the WDT on the MSP432.

```
MAP_WDT_A_holdTimer(); /* Halt watchdog while we set it up. */

/* Set SMCLK to REFO at 128Khz for low frequency mode. */
MAP_CS_setReferenceOscillatorFrequency(CS_REFO_128KHZ);
MAP_CS_initClockSignal(CS_SMCLK, CS_REFOCLK_SELECT,
                       CS_CLOCK_DIVIDER_1);

/* Configure WDT to timeout after 512k iterations of SMCLK, at 128k,
  this will roughly equal 4 seconds.  A soft reset will be performed
  if the watchdog times out. */
MAP_SysCtl_setWDTTimeoutResetType(SYSCTL_SOFT_RESET);
MAP_WDT_A_initWatchdogTimer(WDT_A_CLOCKSOURCE_SMCLK,
                            WDT_A_CLOCKITERATIONS_512K);

MAP_WDT_A_startTimer(); /* Start the watchdog. */

while (1) {
    foo(params);
    MAP_WDT_A_clearTimer(); /* Service the watchdog. */
}
```

If the WDT expires, the following reset options are available:

- *Hard reset.* This resets the processor as well as all peripheral modules that were set up or modified by the application. It does not reboot the device but returns control to the user code. Contents of on-chip SRAM, that is the volatile memory, are preserved.
- *Soft reset.* Only the execution-related components of the system are reset. The peripheral modules retain their configuration parameters and continue to operate through the soft reset. It does not reboot the device but returns control to the user code. Contents of on-chip SRAM are preserved.

The CCS project called `wdt_a_service_the_dog` contains an example program, adapted from the TI SDK.

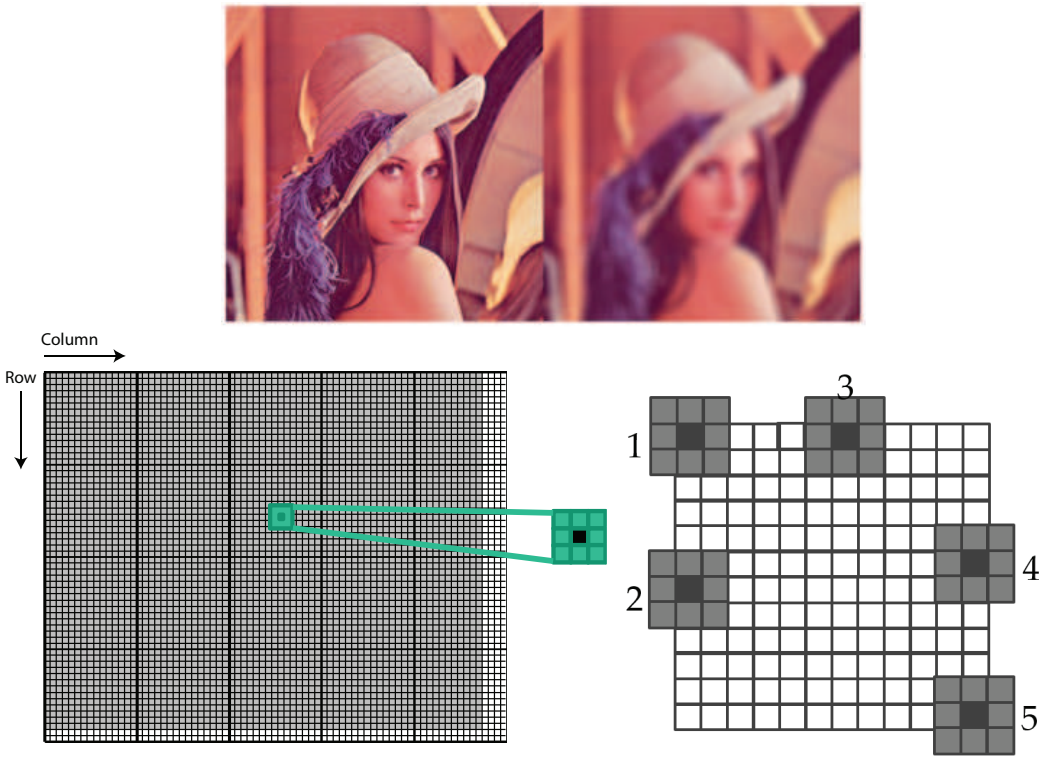

Fig. 1: Illustration of the blurring process along with some examples of boundary cases.

You are supplied with a CCS skeleton project called `program_monitor` containing a function to be monitored using the WDT. The function implements a *box blur filter,* a spatial domain linear filter in which each pixel in the processed image has a value equal to the average value of its neighboring pixels in the input image. Therefore, it is a form of a low-pass or a "blurring" filter. In matrix form, a 3×3 box blur can be written as

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}.$$

Figure 1 illustrates the blurring process. For an interior pixel, the blur value is simply the average of nine values: the eight surrounding pixels as well as itself. For pixels residing along the boundary, for example, cases 1 and 5, the average is calculated over four values, and in cases 2, 3, and 4, the average is calculated over six values.

You may think of the box filter code as being part of a real-time video processing pipeline in which images or frames are acquired from a camera and processed for scene or object detection. The objective of this real-time system is to maintain some desired processing rate in terms of frames per second. We wish to monitor the performance of this system continuously and reset the pipeline if it violates the timing requirements due to a hardware or software problem.

The main processing loop in the code provided to you simulates the acquisition of an image—by generating a `SIZE` $\times$ `SIZE` array called `in` which is populated with random floating-point numbers between `MIN_VALUE` and `MAX_VALUE`. This input array is then processed by the blur filter to provide the array `out`.

```
while (1)
{
    acquireImage(&in, MIN_VALUE, MAX_VALUE, SIZE); // Acquire image
    blurFilter(&in, &out); // Process image
}
```

Answer the following questions:

- **(5 points)** Set the processor to operate at 12MHz and profile the main processing loop to determine WCET for a 75×75 input image. Use the `tic()` and `toc()` functions that you have previously developed for an earlier assignment for this purpose.
- **(10 points)** Once you know WCET, configure and add a WDT to the code that monitors execution of the main processing loop. You may configure the WDT to expire just a tad later than the execution time incurred by the loop. You may drive the WDT using either SMCLK or ACLK sources. Show that the WDT operates correctly during normal operating conditions, as long as it is periodically serviced by the program.
- **(5 points)** Simulate a fault in the main processing loop by inserting a delay. The user should be able to "inject" the fault at any time during program execution by pressing BUTTON1 on the board. Show that the watchdog detects the resulting timing violation and resets the program. If the watchdog just reset the program, toggle LED1 to illustrate that the watchdog timed out. Read through the code example in `wdt_a_service_the_dog` for ideas.

Submit a brief report, up two pages, detailing how you obtained the WCET and how the WDT is integrated within the original code.

Submission Instructions

Once you have implemented all of the required features, submit your code and report by doing the following:

- Run `Clean Project` to remove the executable and object files from the project folder. We must be able to build your projects from source and we don't require your pre-compiled executables or intermediate object files. **If your code does not at the very least compile, you will receive a zero.**
- Zip up the project and upload the zip file using the Blackboard Learn submission link found on the course website.
- Upload the report as a separate PDF document.